FEATURE ARTICLE

Jan van de Kamer

USB Parallel Port

With each passing generation of PC and OS technology, it's becoming harder for you to connect your home-built hardware to your computer. However, with Jan's USB-to-standard parallel port adapter, you will have no problem connecting your own hardware.

Life is becoming more and more difficult for the electronic enthusiast who wants to connect his own hardware to a PC. In the old days, a handful of standard logic chips, a parallel printer port, and a simple DOS program would do the job. Nowadays you need knowledge of microcontrollers, Windows device drivers, and hardware protocols like USB. Windows 98 still allows direct access to the registers on the printer port hardware, but newer versions of Windows such as Windows 2000 strictly forbid this. Even worse, there are plans to declare parallel and serial ports obsolete!

Of course, it is always possible to resist the new generations of PC hardware and operating systems by using the Pentium 233-MHz machine running Windows 95. But eventually there will come a time when new software releases of your favorite CAD systems will not run on the old machine. Sure, a new PC is quickly bought, but what do you do with your hardware projects that took many hours of time to build?

One possible solution is a USB-to-standard parallel port adapter that doesn't use a microcontroller or other programmable logic. Although there is no need to develop a Windows device driver for this adapter, it is unfortunately necessary to modify your project software. There is no way around this. On the bright side, you get something back for this effort: now multiple adapters can be connected to the PC simultaneously. Note also that this adapter cannot be used to drive an actual printer with standard printer drivers; the reasons for this will become clear shortly.

PARALLEL PORT EMULATION

If you ignore bidirectional modes (e.g., EPP and ECP), a standard parallel port has 12 outputs and five inputs that all operate at TTL levels, as shown in Figure 1. Reproducing this pinout is not difficult. Two 74HCT273 chips provide 16 outputs, and eight inputs can be generated by using a 74HCT541. The problem now becomes a question of connecting the resulting 8-bit bus to the serial USB interface. It seems Future Technology Devices International (FTDI) considered this question as well, and its answer was to develop the versatile FT8U245AM USB-to-parallel interface chip.

The FT8U245AM is a full USB V.1.1-to-parallel converter that contains all of the logic required to identify itself as a USB device, to initialize the interface, and handle all USB communication. Your project hardware has two buffers that it can access, one of 384 bytes for sending data to the PC and one of 128 bytes that contains the data received from the PC. As a bonus, FTDI supplies a free Windows device driver that can be downloaded from the company's web site.

Unfortunately, the FT8U245AM is available only in a QFP package. There are no sockets for this package and it's difficult to solder by hand. However, it can be done and you need to build the interface only once. Figure 2 lists the pinout of the chip. I'll discuss the main signals, but for the full details, download the datasheet from the FTDI web site.

USB OPERATION

The USB hardware and driver software on the PC work together to create virtual bidirectional pipelines for data flowing between an application program and a particular USB device. When an applica-[illegible]n writes a byte via the driv-[illegible] API, that byte eventually appears in the receive FIFO of the FT8U245AM. Similarly, when a byte is written by the hardware to the chip's transmit FIFO, it becomes available to the software a short time later.

This project takes advantage of that in the simplest possible way by requiring that the application send data 2 bytes at a time, representing the 16 outputs. For every 2 bytes sent, the hardware returns 1 byte representing the eight inputs. This means that the software must always execute a read function after every 2-byte write function in order to prevent the incoming pipeline from overflowing. These simple rules mean that the hardware can be nothing more than a simple state machine that generates a series of signals that will always be the same. A GAL or PAL would be ideal for this job, but either [illegible]ould require access to a specialized programmer, and I want to avoid that.

Figure 3 shows the order in which the different signals need to be generated to latch the output data into two 74HCT273 8-bit latches for the 16 outputs and to read one 74HCT541 buffer for the eight inputs. The *RXF signal from the FT8U245AM is used as a trigger for the state machine. The first step is to lower the *RD signal releasing the first byte for the outputs, which are clocked into the first 74HCT273 by pulsing its clock signal (CP1). Both CP1 and *RD can be unasserted at the same time.

Another lowering of *RD releases the second byte of outputs that get clocked in the second 74HCT273 by a pulse on CP2. After unasserting *RD the second time, the *RXF signal should be at a high level because all of the data has been read from the receive buffer. The next step is to drive the inputs onto the data bus by lowering the *OE signal of the 74HCT541, clocking the current state of the inputs into the transmit buffer of the FT8U245AM on the falling edge of the WR signal. When this sequence is completed, the state machine will stop in state 0 until new data is received.

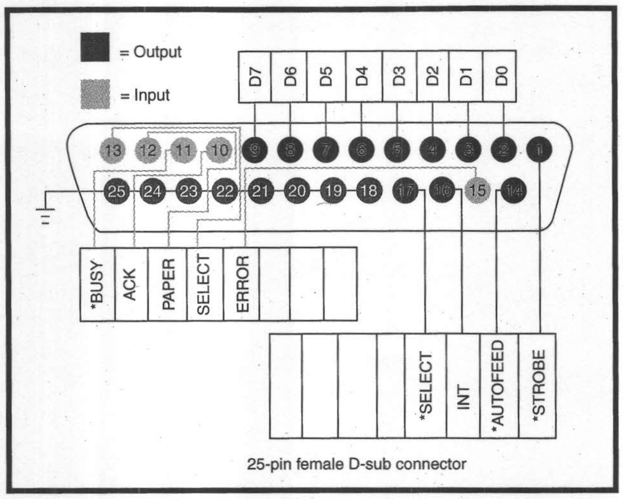


Figure 1—*The standard PC printer port connector is mapped to the control registers of the PC hardware.*

THE SCHEMATICS

Figure 4 presents the schematics for my USB Parallel Port (see Photo 1), including the FT8U245AM (U1) along with the input and output buffers U3 through U5. The resistor array R9 makes sure the inputs are always well-defined even if they're not used. Connector CN2 is a 25-pin D-sub connector identical to those found on the rear of a PC. It carries 12 of the outputs and five of the inputs. The remaining inputs and outputs are connected to CN3. These will not be used for existing hardware projects, but it would be a shame not to make them available for future projects.

The heart of the state machine is U6, a 74HCT4017 10-state counter with fully decoded outputs. R10, C4, and a NAND Schmitt trigger create an oscillator for this counter. It starts only when there is data in the receive buffer of the FT8U245AM (*RXF = 0) and runs until the state machine is finished processing the data. The NOR gates of U8 and a NAND gate in U7 combine the outputs to generate the signals shown in Figure 3.

Theoretically, there can be a timing problem on the *RD signal. A glitch can be generated as output Q1 of the state machine goes low and Q2 goes high. This could be interpreted by the FT8U245AM as the next read pulse. Fortunately, the parasitic capacitance of the outputs of the 74HCT4017 results in a small overlap of the output signals, and the measurements show no spikes in the decoded signals.

A full cycle of the state machine requires 10 clock pulses. If at that time there is more data in the receive buffer, then a second cycle will follow immediately. This can be used to boost the speed of the interface significantly. The maximum speed of the USB interface is 12 Mbps. And, when the overhead of addresses and checksums are subtracted, this results in an effective data transfer rate of 1 MBps. The FT8U245AM is capable of handling this data rate, so the USB Parallel Port's hardware must be able to keep up. Because two bytes are read per cycle, the state machine must be able to execute 524,288 cycles per second. Each cycle requires 10 clock pulses, so an oscillator frequency of a little less than 6 MHz is sufficient. The values of R10 and C4 are selected to produce this frequency.

To put the outputs into known states after a power-up cycle, a reset circuit is incorporated using R12 through R14, Q1, and C8. A NAND

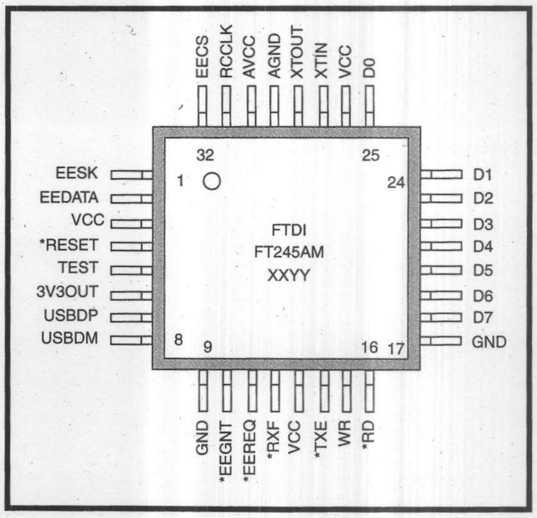


Figure 2—*From the top of the 32-pin Quad Flat Pack package, you can see the pinout of the FT8U245AM.*

age is generated by a low-dropout regulator on the FT8U245AM. Pin 6 provides a 3.3-V power supply that's derived from the voltage on the chip's V_{CC} pins. All other data and control signals of the FT8U245AM use the V_{CC} level, which can be between 4.4 and 5.25 V.

This voltage range is exactly what's available on a standard USB interface, which allows USB hardware to use the

the signal is flank-triggered.

PC's power supply instead of an external power supply. The USB host interface supports an initial current of 100 mA. After the USB device is configured, it can request an increase in current to a maximum level of 500 mA. The host controls this. If there is no power available, the request is denied and the device is shut down.

The maximum output current depends on the input voltage. At 9 VDC, U9 must drop 4 V, which limits the current to 2.6 W divided by 4 V, giving a total of 650 mA. If more current is required, U9 must be mounted standing up and attached to an aluminum heatsink. It's also advisable to build up the ground and power lines to connector CN2 with solder in order to reduce their resistance.

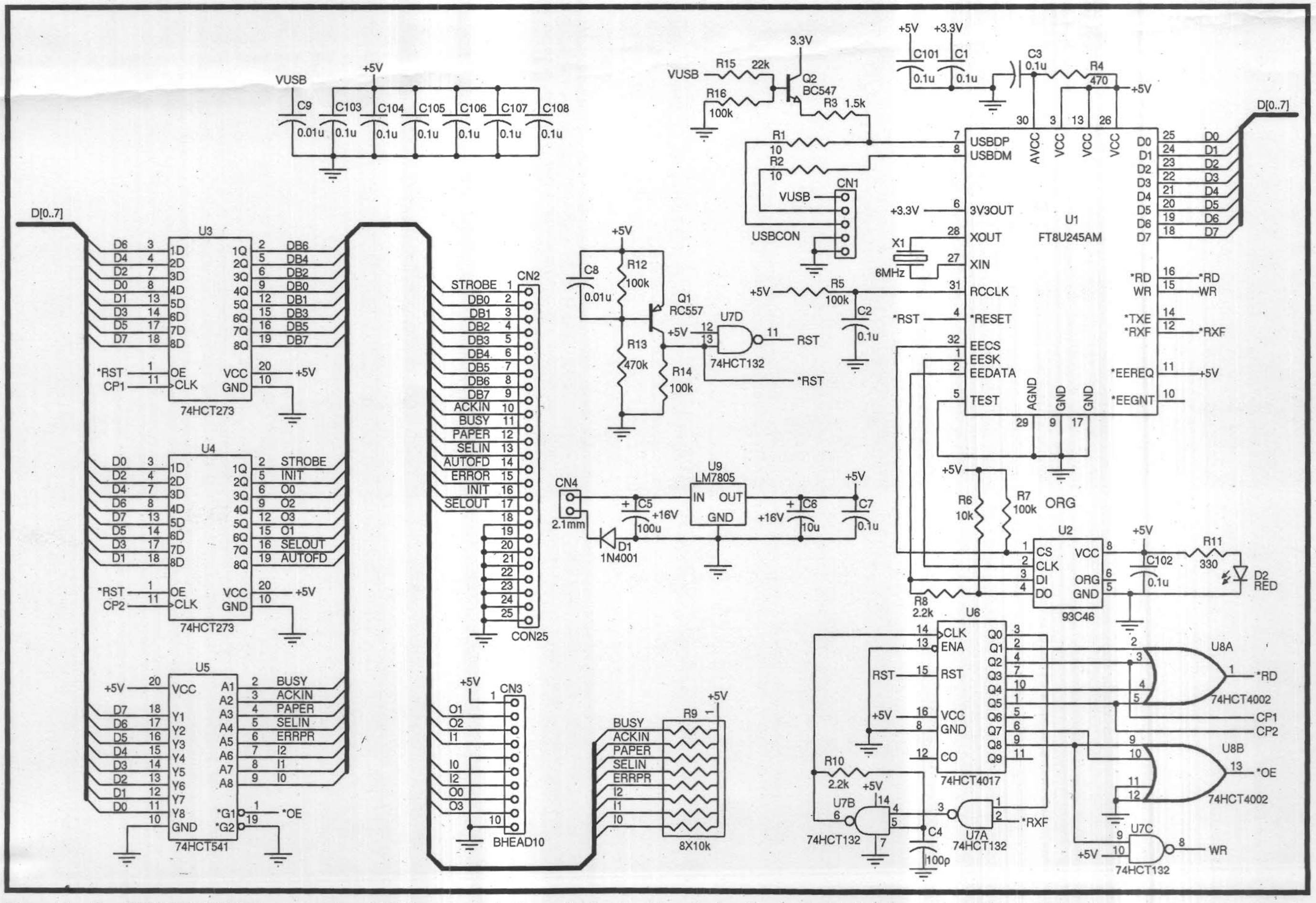


Figure 4—*The FTDI FT8U245AM and standard CMOS series for logic are the basis of the electronics required to build your own USB parallel port.*

In the USB design rules, it states that an externally powered device is not allowed to put voltage on the signal lines if the PC is switched off. This requires the USB Parallel Port to cut off the 3.3 VDC generated by the FT8U245AM because current can flow into the interface via pull-up resistor R3. To prevent this, transistor Q2 in combination with resistors R15 and R16 block the 3.3-VDC power supply to R3. As soon as the PC is switched on, the DATA+ line is pulled to 3.3 VDC, indicating to the PC that this is a high-speed USB device (i.e., supporting data transfers at 12 Mbps). If R3 were connected to the DATA– line, the PC would limit the data transfer speed to 1.5 Mbps. The FT8U245AM adjusts itself accordingly.

THE FIRST TIME

After the PCB is finished, I imagine you'll want to turn on the power. Don't do it just yet; first, mount the PCB onto a flat surface or in a housing. It wouldn't be the first time a small piece of wire caused a short on

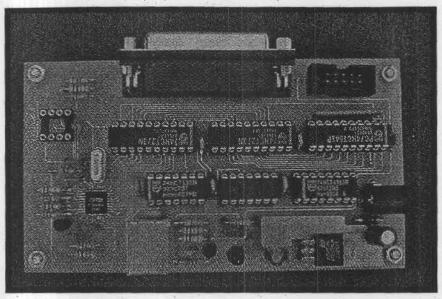


Photo 1—*I made the prototype of the USB Parallel Port by hand.*

your PCB when you just placed it on a table. Secondly, you need to do some work on the PC before you can connect the USB Parallel Port.

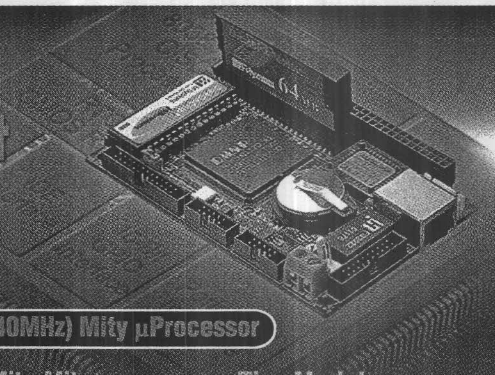
I have developed a special test program for the USB Parallel Port that you may download from my web site (home.planet.nl/~jkamer) or the *Circuit Cellar* ftp site. The installation program Setup.exe takes care of multiple tasks. The test software is installed as well as the EEPROM programmer software from FTDI. The device drivers for the FT8U245AM are placed in a FTDI subfolder, which are needed only when the USB Parallel Port is connected for the first time. A documentation sub-folder that contains several specs and user manuals is also created. Finally, the source files of the test program are placed in a source folder. The test program is developed using Delphi. The best part is that the software is free and can be distributed freely.

After installing the test software, you are all set and ready to go. Now, connect a 9-VDC power supply to CN4. LED D2 will light up to indicate that power is available. This is the moment you've been waiting for: it's time to plug a USB cable type AB into your PC and into the USB Parallel Port. Windows will start thinking (the famous hourglass) and prompt you with a "New Hardware Found" dialog.

The rest is simple as you follow the instructions presented by Windows. As soon as Windows requests the location of the device information file, point it toward the folder that contains the FTD2XX.INF file. After completion of the installation, the status of the FT8U245AM can be requested using the Windows device manager

DM&P Group

Act Mighty, Think Compact . . . Mity µProcessor



Racing up fast with our dynamic Microprocessor - DMP6117D (386SX-40MHz) Mity µProcessor

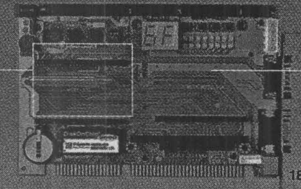
Mity-SoC Series

Mity-SoC Module

65 x 45 mm

Mity-Mite VGA/LCD Development Kit

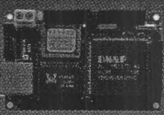
ISA-bus x86 Development Board


185x122mm

Features
- Intel 386SX compatible kernel
- Single Power Source,+5 V@400 mA
- Low Power, Low Heat (Fanless)
- Cold boot in -20°C environment
- Quick-boot BIOS (less than 4 seconds)
- Wide Operating Temperature range -40°C ~ +80°C
- Linux O/S Support
- MS DOS compatible
- Complete TCP/IP Support for DOS - DSock (BOOTP/DHCP, FTP, HTTP, SMTP, TELNET, Talk, etc...)
- Provide Graphic LCM, Serial Port, DOS XMS, and DOS AccTimer Library

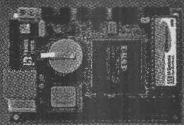
Mity-Mite

Mity-Mite Module


80 x 50 mm

Tiny Module

ICOP-6015


100 x 66 mm

DOS Socket Library - www.dmp.com.tw/tech/dsock

Graphic LCM DOS Library - www.dmp.com.tw/tech/lcm

Serial Port DOS Library - www.dmp.com.tw/app/rsip

DOS XMS Library - www.dmp.com.tw/tech/xms

Contact your regional sales office for Available Development Kits - get a developer package of the Mity series products with our DSock Library and Other Software Utilities to complete your embedded design. We provide the Mity-Mite Module Demo Box to test from browser, send/receive mail and accept FTP download/upload.

Support X-Linux and DSock Library Kernel

ICOP
intelligent control on processor

ICOP TECHNOLOGY INC.

Taiwan	Tel: 886-2-8990-1933	Fax: 886-2-8990-2045	E-mail: info@icop.com.tw
America	Tel: 1-(626)-444-6666	Fax: 1-(626) 444-9966	E-mail: info@icoptech.com

www.icoptech.com

(under the Control Panel). A new USB device will appear in the list with the description "FTDI FT8U2xx Device."

What might look strange is that the properties of this device indicate a USB-to-serial device. This is because FTDI also supplies a serial version of its chip, the FT8U232AM. The device driver for both chips is the same; the parallel version uses only a subset of the functions. For example, commands that set the data rate are not needed for the parallel version.

If Windows doesn't find the new hardware or does not recognize the hardware as a USB device, you face an interesting challenge. But, don't throw the PCB out the window just yet; first, remove the USB cable and power supply. If you didn't do it yet, use a multimeter to check that there are no shorts between the pins of the FT8U245AM. Then, measure all of the tracks among the components to make sure the PCB is OK. You can also reconnect the power and measure the voltage on the logic pins; these should be at a nice high or low logic level. If everything checks out and after reconnecting it to the PC it still is not recognized, your only remaining option is to try a different PC. My PC is one of the first that contained USB ports and Windows 98SE as an operating system. It seems that the first USB host chips were not as compliant with the standard as they claimed. Replacing the USB card with a later model solved the problem.

After Windows has installed the FTDI device drivers, the test program can be launched. After detecting the USB Parallel Port, all of the outputs can be modified directly using the checkbox closest to the output you want to change. Photo 2 shows a screen dump of the software. The checkboxes that are placed close to the inputs cannot be modified but instead display the current status of the corresponding input. A mark in the checkbox indicates a high level, and no mark means a low level. Note that if you change the level at one of the inputs on the hardware, then you will not see this in the software until you change an output.

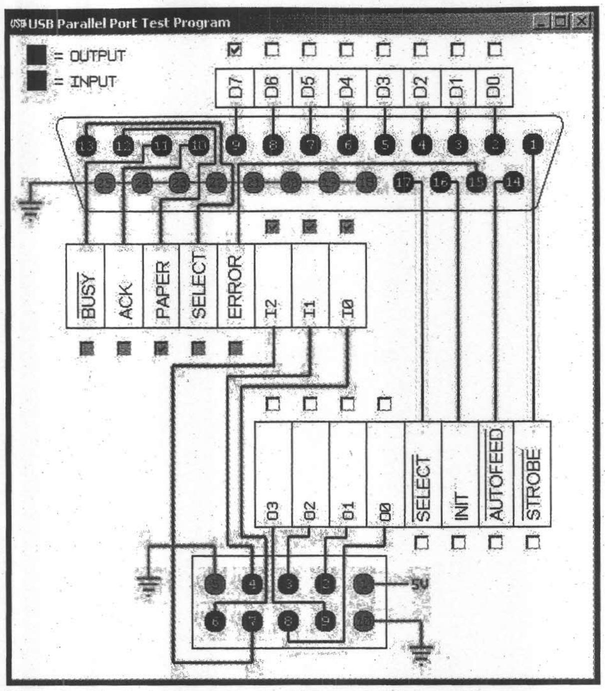


Photo 2—*The test software for the USB Parallel Port provides easy access to each input and output.*

DEVICE CONFIGURATION

I succeeded in designing the USB Parallel Port without the use of microcontrollers or any other programmable hardware. However, a small EEPROM memory is connected to the FT8U245AM at U2. Proper operation of the USB Parallel Port does not require this chip, so why put it there?

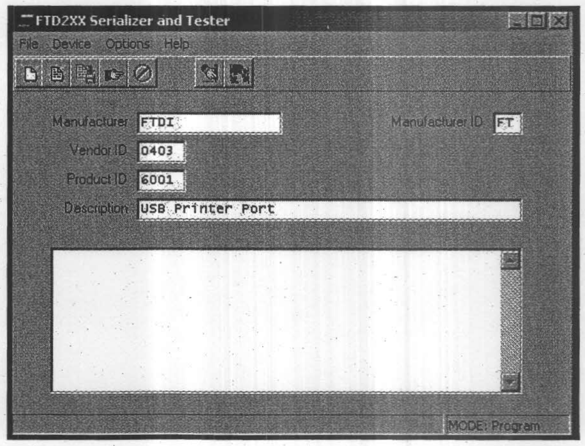
Every USB device must identify itself to the PC; this is done by two codes, the Vendor ID (VID) and the Product ID (PID). These two 16-bit numbers are the first the PC calls for after a device is connected. Depending on the value of these numbers, Windows will decide which device drivers must be loaded to support the device. If the device is a known device, Windows takes care of this without your help. If the device is new, Windows will prompt you with the "New Hardware" dialog.

It is possible to connect more than one of the same type of device. In order to tell them apart, each device must have its own serial number and device description. By default the FT8U245AM uses a VID of $0403, a PID of $6001, no serial number, and no device description. Because no serial number or device description is given, by default only one FT8U245AM device can be connected to the PC.

With the help of an EEPROM these values can be changed to make the devices unique. You can request a new VID number from the USB organization, and then change the FTDI's default value. This isn't affordable for most hobbyists, so instead you might want to limit yourself to changing the serial number and device description.

The FT8U245AM spec documents how to organize the EEPROM contents. However, it is not necessary to develop an EEPROM programmer; the FT8U245AM can take care of this itself using the USB interface. To make life easy, FTDI provides a program that allows you to enter the settings you want (see Photo 3). You can download the program from the FTDI web site.

There are many suppliers of serial EEPROMs. The basic function is the same but the use of non-serial pins 6 and 7 and the memory layout seem to vary from supplier to supplier. The chip used here must have a 64 × 16 bit memory layout, and FTDI recommends the 93LC46B (among others)



Manufacturer FTDI
Manufacturer ID FT
Vendor ID 0403
Product ID 6001
Description USB Printer Port

Photo 3—*The EEPROM programming utility for programming the EEPROM code in-system was developed by FTDI.*

from Microchip. Available in many packages, I selected a 93LC46B-P, the P indicating an 8-pin DIL package.

Place an empty EEPROM in the socket for U2 after removing the USB cable and switching off the power supply. After inserting the EEPROM, the USB Parallel Port can be connected to the PC again and the EEPROM configuration program FTD2XXST.EXE can be launched. Fill in the requested data using the VID and PID numbers from FTDI. If you changed these numbers, Windows will recognize the parallel port as a different device because it won't know the new numbers. The FTD2XX.INF configuration file must be changed to reflect the new values. The device drivers themselves do not have to change.

After entering the required data in FTD2XXST.EXE, it can be written into the EEPROM by selecting the Program command from the Device menu. Remove the USB cable, switch off the power supply, and then switch on and insert the USB cable again. Windows will find new hardware but the PID and VID will be known this time, so Windows will assume the device uses the registered device drivers.

From this point onward, you'll have the ability to connect multiple USB Parallel Ports. The test software does not support this, but because the source code is available (including a special Delphi USB Parallel Port object), it's easy to modify the software as needed. Note that you still have to do some of the programming yourself.

PERFORMANCE MEASUREMENT

After every write cycle of 2 bytes, the hardware places 1 byte in the transmit buffer. The size of the transmit buffer is 384 bytes, so after sending 2 × 384 = 768 bytes, the PC must execute a read cycle to prevent the loss of data.

When I tested this on the prototype, I found a strange effect. Even when I transmitted a block of 1000 bytes, I was able to execute a read cycle of 500 bytes. How was this possible? The USB protocol divides every second into 1000 time slots. In each slot, data can be exchanged between a PC and the addressed device. If you send 1000 bytes to the FT8U245AM, the Windows device driver from FTDI will transmit this data to the chip using a handshake protocol over several time slots. After sending each packet, the status of the FT8U245AM is reported back with 1 bit indicating whether the transmit buffer contains data. If so, this buffer is read by the PC before sending more data to prevent data loss. This uses up some bandwidth, but the benefit is obvious.

In order to determine the maximum speed of the USB Parallel Port, I connected some of the outputs to inputs. By varying the buffer size and counting how many of these buffers could be processed in 1 s, I produced the graph shown in Figure 5. The x-axis displays the buffer size as passed to the device driver. For example, a value of 1025 indicates a transmit buffer of 2050 bytes and a receive buffer of 1025 bytes. The number of cycles is listed on the y-axis, where one cycle represents setting 16 outputs and reading eight inputs.

The graph shows a maximum of 190,000 changes to the outputs per width is divided and the throughput will decrease accordingly. ■

Jan van de Kamer earned a B.S. in Technical Computer Science from Hogeschool Arnhem, the Netherlands, in 1993. After graduation, he successfully started his own business designing microcontroller embedded systems for industrial applications. On the side, he designs robots using these types of systems. He enjoys interfacing with mechanical parts and bringing them to life. You may reach Jan at jkamer@planet.nl or through his web site at home.planet.nl/~jkamer.

PROJECT FILES

To download the PCB layouts, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2003/151/.

RESOURCE

USB Information
USB Implementers Forum
(503) 296-9892
www.usb.org

SOURCES

FT8U245AM Interface chip
Future Technology Devices International Ltd.
44 141 353 2565
www.ftdichip.com

93LC46B Chip
Microchip Technology Inc.
(480) 786-7200
www.microchip.com